UNIVERSITY OF GLASGOW

Computing Department

MULTUM STANDARD FUNCTION LIBRARY

Memo No. 1
11/5/73.

A standard interface convention for the "mathematical" functions.

1 Aims

These notes define the conventions to be observed in using and implementing the library of "mathematical" functions for the MULTUM. It should be possible to use the same library from Usercode (SUL), FORTRAN, Algol W and Pascal programs. Extensions of these conventions would allow for mixed-language programming of some generality, but that possibility is not treated further here.

2 Standard scalar data types

We must handle data of the following types:

(a) 32 bit integers
(b) 32 bit reals
(c) 64 bit reals
(d) 64 (2 * 32) bit complex
(e) 128 (2 * 64) bit complex
(f) 16 bit integers (Pascal only).

The integer types and the 32 bit reals are defined by the hardware of the ALP/3 processor. The 64 bit reals and the 128 bit complex numbers have not yet been defined, and will be handled entirely by software.

Further types may be added to this list as new languages or new features of the current languages gain the support of the library.

3 Parameter communication

For the functions considered all parameters are passed by value.

4 Standard procedure linkage

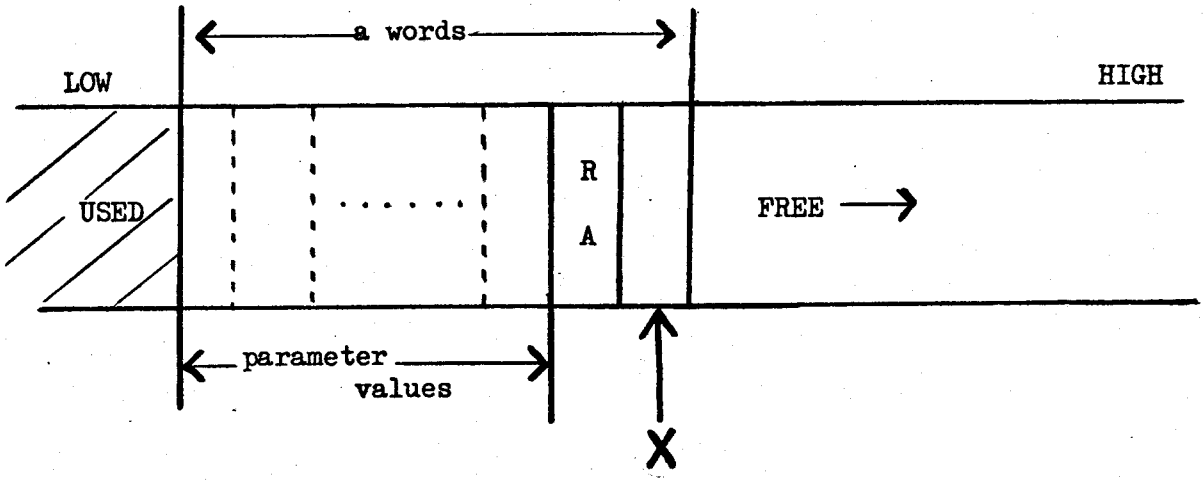
A linkage mechanism of full generality has been defined for procedure calls and a special case of this is used by the library functions. To provide the background to the methods used in the library a description of the general mechanism follows.

Each program has a module integrated into a READ/WRITE segment to act as a stack. Parameters and return addresses are transmitted to a procedure on the stack and the result of a (long) function may be left there. Some languages (e.g. Pascal) use the stack to store the local variables of a procedure and are thus capable of recursion. Others may use the stack merely as a working storage area. In any case a standard situation obtains on entry to a procedure and a standard situation must be set up by it before return.

4.1/

4.1 Entry conditions

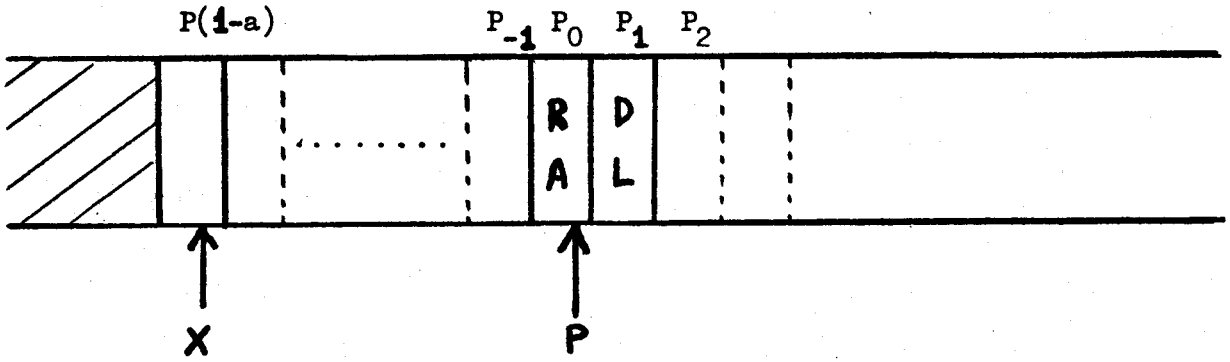
On entry to a procedure the X register must be pointing to the next free word on the stack. No requirement is placed on the contents of any of the other registers. The actual parameters and the return address link are stacked in that order. Thus the following diagram holds:



Here RA denotes the return address link (see 4.3).

4.2 Exit conditions

On return from a procedure the following situation will hold in the stack (DL, denoting dynamic link, is a copy of the value in P on entry to the procedure):



If there are any value result or result parameters they can be/

/be accessed using negative P-relative addresses. If the procedure is a function with a result of 32 bits or less, this will be present in E (or A). Functions with longer results leave them in the locations following the dynamic link. To restore P to the value it had before the procedure call the instruction LODP P1I must be obeyed.

4.3 The return address link

On exit from a procedure return is made to the instruction at ((RA)+1).

5 Macros

A set of macros is available to simplify the programming of the linkage mechanism and access to parameter, result and working locations. These macros should be used, as this guarantees that the standard interface conventions are observed.

5.1 STCK

STCK=<reg>, where <reg>::= A|B|E

generates code to stack the corresponding register. When STCK is obeyed, X must hold the address of the top of the stack. STCK updates X appropriately.

5.2 SPOP

SPOP=<reg>

generates code to unstack a value into the given register. The conditions of use are as for STCK.

5.3 SCFP

SCFP=<name>=<reset> , where
<reset>::= Y|N

generates the code to call a function or procedure whose entry point is given by <name>. If the second parameter is given as Y then code will be produced to reset the P register on return. Otherwise the user must reset P himself (by "LODP P1I").

5.4 SENT

SENT=<name>=<number>=<type> , where
<number>::= <integer>
and <type> ::= 1 | 2 | 4 | 8

generates an entry point for a function or procedure called <name> with <number> parameters, each of length <type>. If several/

/several entry points are defined in succession using this macro, they must have the same parameter specifications.

5.5. SRES

SRES=<name>=<type>

defines the name and length of the result of a function. Any call on SRES must immediately follow a call of SENT.

5.6 SPAR

SPAR=<name>

defines <name> as the label of the next parameter of a function or procedure, which can then be accessed using mode P(<name>). All calls of SPAR must follow SENT or SRES.

5.7 SWRK

SWRK=<workspace>=<mhrs>
where <workspace>::= <mhrs>::= <integer>

generates code to reserve <mhrs> memory held registers (<mhrs>$\leq$16) and <workspace> additional words of working storage for local variables (<mhrs>+<workspace> $\leq$ 128). SWRK, if used, must follow all uses of SPAR.

5.8 SLCL

SLCL=<name>=<type>

defines <name> as the next local variable of the procedure or function. The length of the variable is given by <type>. Following a call of SLCL the variable can be accessed using mode P (<name>). All calls of SLCL must be preceded by a call of SWRK.

5.9 SXIT

SXIT= <result>=<name>

generates the code to effect an exit from a function or procedure.

(a) Procedures: <result> must be given as N and <name> as 0 (zero).

(b) Functions: If the (16- or 32-bit) result should be fetched into a register <result> must be given as Y . Otherwise, <result> should be given as N . <name> must be the label of the result location (as defined using SRES).

5.10/

5.10 Assembler registers

The assembly-time registers (% & $ @) must not be used by programs calling these macros.

6 Example

```
ZMOD CABS
SENT=CABS=2=2
// THE COMPLEX PARAMETER IS TREATED AS TWO REALS
SRES=ABSVAL=2
// THE RESULT IS REAL
SPAR=REALPT
SPAR=IMAGPT
SETE P(REAL PT)
MLTF P(REALPT)
STES P(ABSVAL)  / USING RESULT LOCN. AS TEMP.
SETE P(IMAGPT)
MLTF P(IMAGPT)
ADDF P(ABSVAL)  / SUM OF SQUARES IN E
STCK=E
SCFP=SQRT=Y
// STACK A PARAMETER FOR, THEN CALL, SQRT
SXIT=N=ABSVAL
// THE RESULT IS ALREADY IN E
ZEND
```